Final Design Document

Among them the main ones are Grid, Block, and Level. One of us – Samir Musali was responsible for writing implementation of Level and its subclasses, and another – Rasul Rasulzada was responsible for writing implementation of TextDisplay, XWindow, and Cell. We wrote implementation of others – Grid and Block together. The game is established on Grid. Grid has 2 Block pointers: currentBlock and nextBlock. Blocks themselves consist of 4 Cells regardless of their types. Blocks have their own identification character which will differ one from another according to their type. The movement command functions are defined in the Block except rotate functions, and in addition to identification character every Block has a colour. Default level is set as Level 0; so, the game starts within Level 0, and the next blocks will be generated according to the given file; so, if there is no file, Level 0 takes "sequence.txt". In the beginning of the game firstly getNextBlock is called, it produces our first current block on the Grid, and then we call it again; so, the game is ready to start. After "Drop" command is executed, currentBlock is updated and getNextBlock will be called again to generate new Block. The way of generating blocks changes by the Level. If player wants to change Level, the game will be go on, but the method of generating Blocks will be changed. The block showing as next still comes next, but subsequent blocks are generated using the new level. If the game is restarted, the all in Grid except hiScore is recreated. We can see that "Down" is a kind of "Drop"; in other words, "Drop" consists of finite number of "Down" functions; so, after each dropping, Grid will check its lines in which line is full or not; if any line is full, it will be cleared, and everything above comes down. Each Cell has its isFree field which shows it is active (or occupied) or not. When nextBlock replaces the currentBlock, the Cells of nextBlock will be appeared on Grid. We know that moving Block means moving its Cells; so, when the Block changes its coordinates, the Cells in the old coordinates will be freed and the new ones will be occupied. We used the method DrawBlock for drawing it under the game board to show which Block is next. We used TextDisplay and Xwindow similarly as we used in Assignment 4. In addition, we want to add extra feature called "skip" in the end of implementation; so, this new feature skips the currentBlock, currentBlock will be replaced by nextBlock, and new nextBlock will be generated. We have also used trie.h library from the Assignment 3 in the main.cpp for reading shortened commands. We have also used the special design for graphical display, as extra feature.

Final Design Document

QUESTION1:

How could you design your system to make sure that only these seven kinds of blocks can be generated, and in particular, that non-standard configurations (where, for example, the four pieces are not adjacent) cannot be produced?

ANSWER:

We decided to initialize each block according to the given character that is one of I,J,S,Z,T,O or L. Hence, if something else is given, it will not create any block and none of the “Cell”s will be occupied.

QUESTION2:

How could you design your system (or modify your existing design) to allow for some generated blocks to disappear from the screen if not cleared before 10 more blocks have fallen? Could the generation of such blocks be easily confined to more advanced levels?

ANSWER:

Since Blocks consist of Cells, and each Cell has the field called isFree and describes whether it is occupied, for Blocks to disappear we will set each of Blocks' Cells free and call notifyDisplay. If you meant how Blocks will disappear while clearing full line, we will call removeline method, and since we do not keep track of dropped Blocks, removeline method is not about Blocks, it is about Cells.

QUESTION 3:

How could you design your program to accommodate the possibility of introducing additional levels into the system, with minimum recompilation?

ANSWER:

Actually we have a superclass named “Level” and 4 subclasses for each level, so we would create a new subclass for addition level.

QUESTION4:

How could you design your system to accommodate the addition of new command names, or changes to existing command names, with minimal changes to source and minimal recompilation? (We acknowledge, of course, that adding a new command probably means adding a new feature, which can mean adding a non-trivial amount of code.) How difficult would it be to adapt your system to support a command whereby a user could rename existing commands (e.g. something like rename counter-clockwise cc)?

ANSWER:

If new command is added, we would create a new class with definition and declaration of that command and then we can easily “connect” it to the main source code with “Visitor Pattern”. That will not change other classes and game will accept it.

QUESTION5: (new)

What lessons did this project teach you about developing software in teams? If you worked alone, what lessons did you learn about writing large programs?

ANSWER:

We have learned working with classes using different patterns. We have learned how to write efficient codes, handle the memory leaks, and work on graphical display.

Final Design Document

QUESTION6: (new)

What would you have done differently if you had the chance to start over?

ANSWER:

If we had the chance to start over, we would add two extra functions: one of them substitutes the NextBlock with the CurrentBlock, and another one finds out the appropriate place for the CurrentBlock to drop; also we would like to write the program using ncurses library for convenience.